hoodie-test Documentation

Release stable

February 27, 2017

Guides

CHAPTER 1

Welcome to Hoodie

Hoodie is a backend for web applications with a JavaScript API for your frontend. If you love building apps with HTML, CSS and JavaScript or a frontend framework, but *dread* backend work, Hoodie is for you.

Hoodie's frontend API gives your code superpowers by allowing you to do things that usually only a backend can do (user accounts, emails, payments, etc.).

All of Hoodie is accessible through a simple script include, just like jQuery or lodash:

```
<script src="/hoodie/client.js"></script>
<script type="javascript">
  var hoodie = new Hoodie();
</script>
```

From that point on, things get really powerful really quickly:

```
// In your front-end code:
hoodie.ready.then(function () {
  hoodie.account.signUp({
    username: username,
    password: password
  })
})
```

That's how simple signing up a new user is, for example. But anyway:

Hoodie is a frontend abstraction of a generic backend web service. As such, it is agnostic to your choice of frontend application framework. For example, you can use jQuery for your web app and Hoodie for your connection to the backend, instead of raw jQuery.ajax. You could also use React with Hoodie as a data store, or any other frontend framework or library, really.

Open Source

Hoodie is an Open Source project, so we don't own it, can't sell it, and it won't suddenly vanish because we got aquired. The source code for Hoodie is available on GitHub under the Apache License 2.0.

How to proceed

You could read up on some of the ideological concepts behind Hoodie, such as noBackend and Offline First. These explain why Hoodie exists and why it looks and works the way it does.

If you're more interested in the technical details of Hoodie, check out How Hoodie Works. Learn how Hoodie handles data storage, does syncing, and where the offline support comes from.

Eager to build stuff? Skip ahead to the quickstart guide!

CHAPTER 2

Quickstart

In this guide you'll learn how to create a demo Hoodie app, learn about the basic structure of a Hoodie project and its folders, the endpoints and app URLs and how to include and use the Hoodie library in your project.

Prerequisites

For all operating systems, you'll need Node.js installed. You can download Node from nodejs.org. We recommend the LTS (Long Term Support) version.

Make sure you have version 4 or higher. You can find out with

```
$ node -v
```

Create a new Hoodie Backend

First you need to create a new folder, let's call it **testapp**

```
$ mkdir testapp
```

Change into the `testapp` directory.

```
$ cd testapp
```

Now we need to create a **package.json** file. For that we can use npm which comes with Node by default. It will ask you a few questions, you can simply press enter to leave the default values.

```
$ npm init
```

Now we can install **hoodie** using npm

```
$ npm install hoodie --save
```

The resulting **package.json** file in the current folder, should look something like this

```
{
  "name": "funky",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "hoodie",
```

```
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Now you can start Hoodie with

```
$ npm start
```

Great, your Hoodie backend started up and is now telling you at which URL you can access it. By default that is http://127.0.0.1:8080

Congratulations, you just created your first Hoodie Backend :) You can now load the Hoodie client on any website with

```
<script src="http://127.0.0.1:8080/hoodie/client.js"></script>
```

You can also create a `public/index.html` file, which will be served at http://127.0.0.1:8080 after you restart the server. All assets in the public folder, like images, CSS files or JavaScript files, will be served by your Hoodie Backend at `http://127.0.0.1:8080/<path/to/your/file.ext>`.

What's next?

Our Hoodie Tracker App is a great place to see how to use a Hoodie backend. It's an intentionally simple and well commented application built with only HTML, JavaScript and CSS, without using any library or framework. You can see it running at https://tracker.hood.ie/

Having Trouble?

Sorry it didn't go smoothly for you. Come chat with us or ask a question on StackOverflow

CHAPTER 3

Configuration

Your Hoodie backend can be configured using default options that are part of your repository as well as using hidden files, CLI arguments and environment variables.

Options

Here is a list of all available options

Option	Default value	CLI argument	ENV variable	description
address	`'127.0.0.1'`	`--address`	`hoodie_address`	Address to which Hoodie binds
data	`'.hoodie'`	`--data`	`hoodie_data`	Data path
dbUrl	–	`--dbUrl`	`hoodie_dbUrl`	If provided, uses external CouchDB. URL has to
dbAdapter	`'pouchdb-adapter-dbAdapter`		`hoodie_dbAdapter`	Sets default *PouchDB adapter <https://pou*
loglevel	`'warn'`	`--loglevel`	`hoodie_loglevel`	One of: silent, error, warn, http, info, verbos
inMemory	`false`	`-m`, `--inMemory`	`hoodie_inMemory`	Whether to start the PouchDB Server in mem
port	`8080`	`--port`	`hoodie_port`	Port-number to run the Hoodie App on
public	`'public'`	`--public`	`hoodie_public`	path to static assets
url	•	`--url`	`hoodie_url`	Optional: external URL at which Hoodie Server is a
adminPassword	•	`--adminPassword`	`hoodie_AdminPassword`	Password to login to Admin Dashboard. Login is not possible unless set

Defaults

Default options are set in your app's `package.json` file, using the `"hoodie"` key. Here is an example with all avaialble options and their default values

```
{
  "hoodie": {
    "address": "127.0.0.1",
    "port": 8080,
    "data": ".hoodie",
    "public": "public",
    "dbUrl": "",
```

```
    "dbAdapter": "pouchdb-adapter-fs",
    "inMemory": false,
    "loglevel": "warn",
    "url": "",
    "adminPassword": ""
  }
}
```

.hoodierc

The `.hoodierc` can be used to set configuration when running your Hoodie backend in that folder. It should not be comitted to your repository.

The content can be in JSON or INI format. See the rc package on npm for more information

CLI arguments and environment variables

To pass CLI options when starting Hoodie, you have to separate them with `--`, for example:

```
$ npm start -- --port=8090 --inMemory
```

All environment varibales are prefixed with `hoodie_`. So to set the port to `8090` and to start Hoodie in memory mode, you have to

- set the `hoodie_port` environment variable to `8090`
- set the `hoodie_inMemory` environment variable to `true`

Hoodie CLI is using rc for configuration, so the same options can be set with environment variables and config files. Environment variables are prefixed with `hoodie_`.

The priority of configuration

1. command line arguments
2. Environment variables
3. `.hoodierc` files
4. Your app's defaults form the `"hoodie"` key in `"package.json"`
5. Hoodie's default values as shown in table above

CHAPTER 4

Plugins

You can extend your Hoodie backend in two ways

1. App-specific plugins
2. 3rd party plugins

App-specific plugins

You can extend your Hoodie's client by creating the file `hoodie/client/index.js` in your app's repository, which should export a *Hoodie Client plugin <https://github.com/hoodiehq/hoodie-client#hoodieplugin>*. It will dynamically be bundled into your client accessible at the `/hoodie/client.js` route.

You can extend your Hoodie's server routes and API by creating `hoodie/server/index.js` in your app's, which should export a *hapi plugin <https://hapijs.com/tutorials/plugins>*.

3rd party plugins

Hoodie will soon support loading 3rd party plugins from npm packages. You can watch this pull request for updates.

To get an idea how 3rd party plugins will work and look like, have a look at https://github.com/hoodiehq/hoodie-plugin-example

CHAPTER 5

Deployment

Docker

We continuously deploy our Hoodie Tracker App using Docker. You can read about our continuous deployment set at hoodie-app-tracker/deployment.md.

Now

now is a great way to quickly deploy Node.js applications. Unfortunately, now is a read-only file system, so you must either run your app in memory or set an external CouchDB URL.

Add this script to your package.json and you are good to go:

```
"now-start": "hoodie --inMemory",
```

CHAPTER 6

Hoodie API

Hoodie provides two APIs

1. The Hoodie Client API

   The Hoodie Client API is what you load into your web application using a script tag. It connects to your Hoodie Backend's routes

2. The Hoodie Server API

   The Hoodie Server API is used within Hoodie's route handlers and by plugins to manage accounts, data and to securely integrate with 3rd party services.

The Hoodie Client API

hoodie

Introduction

This document describes the functionality of the hoodie base object. It provides a number of helper methods dealing with event handling and connectivity, as well as a unique id generator and a means to set the endpoint which Hoodie communicates with.

Initialisation

The Hoodie Client persists state in the browser, like the current user's id, session or the connection status to the backend. On page load, Hoodie has to load this state from the local store before you can use its APIs. You can use the Promise returned by `hoodie.ready` to wait until all APIs are fully initialised

```
hoodie.ready.then(function () {
  // all hoodie APIs are ready now
})
```

This is work in progress

Please help us make this awesome <3

For the time being, check out hoodie-client's README.

hoodie.account

The account object in the client-side Hoodie API covers all user and authentication-related operations, and enables you to do previously complex operations, such as signing up a new user, with only a few lines of frontend code. Since data in Hoodie is generally bound to a user, it makes sense to familiarise yourself with **account** before you move on to store.

This is work in progress

Please help us make this awesome <3

For the time being, check out hoodie-account-client's README.

hoodie.store

If you want to do anything with data in Hoodie, this is where it happens.

This is work in progress

Please help us make this awesome <3

For the time being, check out hoodie-store-client's README.

hoodie.connectionStatus

This is work in progress

Please help us make this awesome <3

For the time being, check out hoodie-connection-status's README.

hoodie.log

This is work in progress

Please help us make this awesome <3

For the time being, check out hoodie-log's README.

This library, commonly called **Hoodie Client**, is what you'll be working with on the client side. It consists of:

- The Hoodie Client API, which has a couple of useful helpers
- The account API, which lets you do user authentication, such as signing users up, in and out
- The store API, which provides means to store and retrieve data for each individial user
- The connectionStatus API, which provides helpers for connectivity.
- The log API, which provides a nice API for logging all the things

The Hoodie Server API

The Hoodie Server API is currently work-in-progress. But you can have a look at the Account Server API and the Store Server API for a sneak peak.

CHAPTER 7

Hoodie's Concepts

Hoodie was designed around a few core beliefs and concepts, and they explain a lot if the choices made in the code and the functionality. They are:

- *Dreamcode*
- *noBackend*
- *Offline First*

Dreamcode

While designing Hoodie's API, we realised that we wanted to do more than simply expose some server code to the frontend. **We wanted to reduce complexity, not move it around**. And to make something simple and intuitive, you can't start with the tech stack, you have to start with the humans that are going to use it. What would their dream API look like? Dreamcode is essentially user-centered design for APIs.

To put it bluntly: **Hoodie's API is optimized for being awesome**. For being intuitive and accessible. And it's optimized for making the lives of frontend developers as good as possible. It's also an API first: it's a promise - everything else can change or is replaceable. The API is all that matters.

Forget all the constraints of today's browsers. Then write down the code of your dreams for all the tasks you need to build your app. The implementation behind the API doesn't matter, it can be simple or tough as nails, but crucially: the users shouldn't have to care. This is dreamcode.

Everything is hard until someone makes it easy. We're making web app development easy.

Here's some further information and links to Dreamcode examples.

noBackend

Servers are difficult. Databases are difficult. The interplay between client and server is difficult, there are many moving parts, there are many entertaining mistakes to make, and **the barrier to entry for web app development is, in our mind, needlessly high**. You shouldn't have to be a full stack developer to build a functioning app prototype, or code a small tool for yourself or your team, or launch a simple MVP.

People have been building web apps for quite a while now, and their basic operations (sign up, sign in, sign out, store and retrieve data, etc.) must have been written a million separate times by now. These things really shouldn't be difficult anymore. So we're proposing Hoodie as a noBackend solution. Yes, a backend does exist, but it doesn't have to exist in your head. You don't have to plan it or set it up. You simply don't have to worry about it for those basic

operations, you can do all of them with Hoodie's frontend API. Of yourse, we let you dig as deep as you want, but for the start, you don't have to.

noBackend gives you time to work on the hard problems, the parts of the app that are justifiably difficult and non-abstractable, like the interface, the user experience, the things that make your product what it is.

With Hoodie, you scaffold out your app with

and you're good to go. Sign up users, store data... it's all right there, immediately. It's a backend in a box, empowering frontend developers to build entire apps without thinking about the backend at all. Check out some example Hoodie apps if you'd like to see some code.

More information about noBackend

See nobackend.org, Examples for noBackend solutions and @nobackend on Twitter.

Offline First

We make websites and apps for the web. The whole point is to be online, right? We're online when we build these things, and we generally assume our users to be in a state of permanent connectivity. That state, however, is a myth, and that assumption causes all sorts of problems.

With the stellar rise of mobile computing, we can no longer assume anything about our users' connections. Just as we all had to learn to accept that screens now come in all shapes and sizes, **we'll have to learn that connections can be present or absent, fast or slow, steady or intermittent, free or expensive...** We reacted to the challenge of unknowable screen sizes with Responsive Webdesign and Mobile First, and we will react to the challenge of unknowable connections with Offline First.

Offline First means: build your apps without the assumption of permanent connectivity. Cache data and apps locally. Build interfaces that accomodate the offline state elegantly. Design user interactions that will not break if their train goes into a tunnel. Don't freak out your users with network error messages or frustrate them with inaccessible data. **Offline First apps are faster, more robust, more pleasant to use, and ultimately: more useful.**

More information about Offline First

See offlinefirst.org, on GitHub and discussions and research

So now you know what motivates us

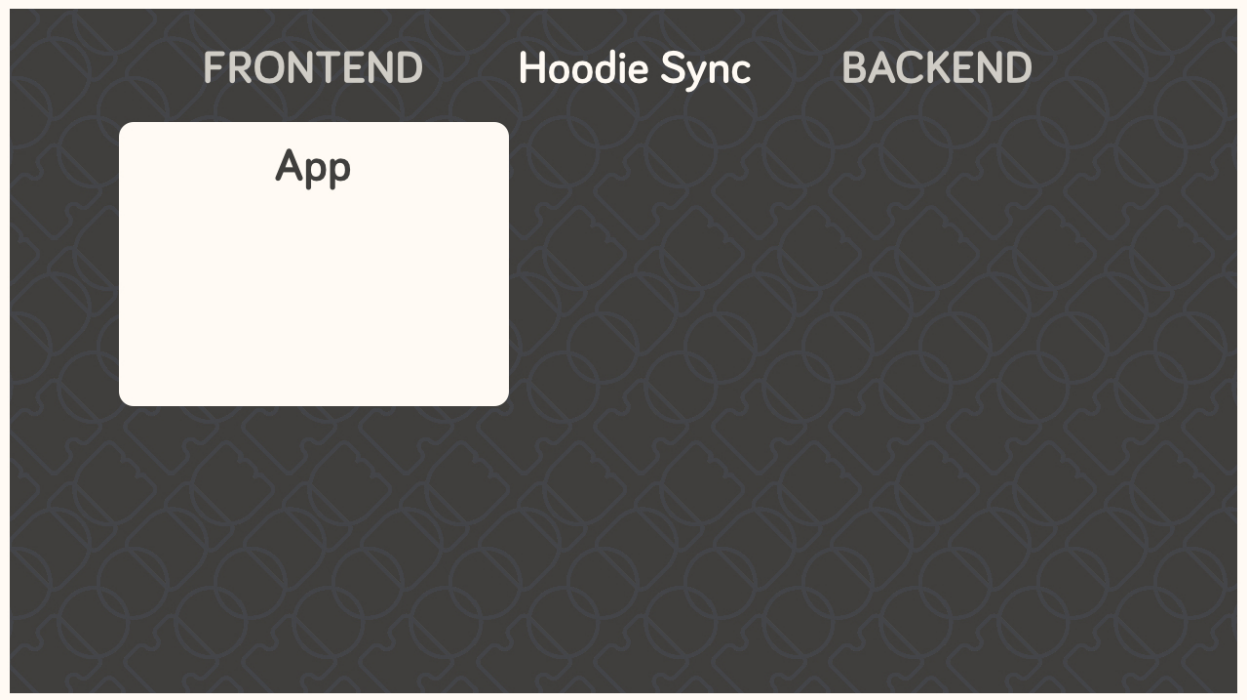
We hope this motivated you too! So let's continue to the system requirements for Hoodie.

CHAPTER 8

How Hoodie Works

Hoodie has several components that work together in a somewhat atypical way to deliver our promise of simplicity, out-of-the-box syncing, and offline capability.

Everything starts in the frontend, with your app. This is your user interface, your client side business logic, etc.



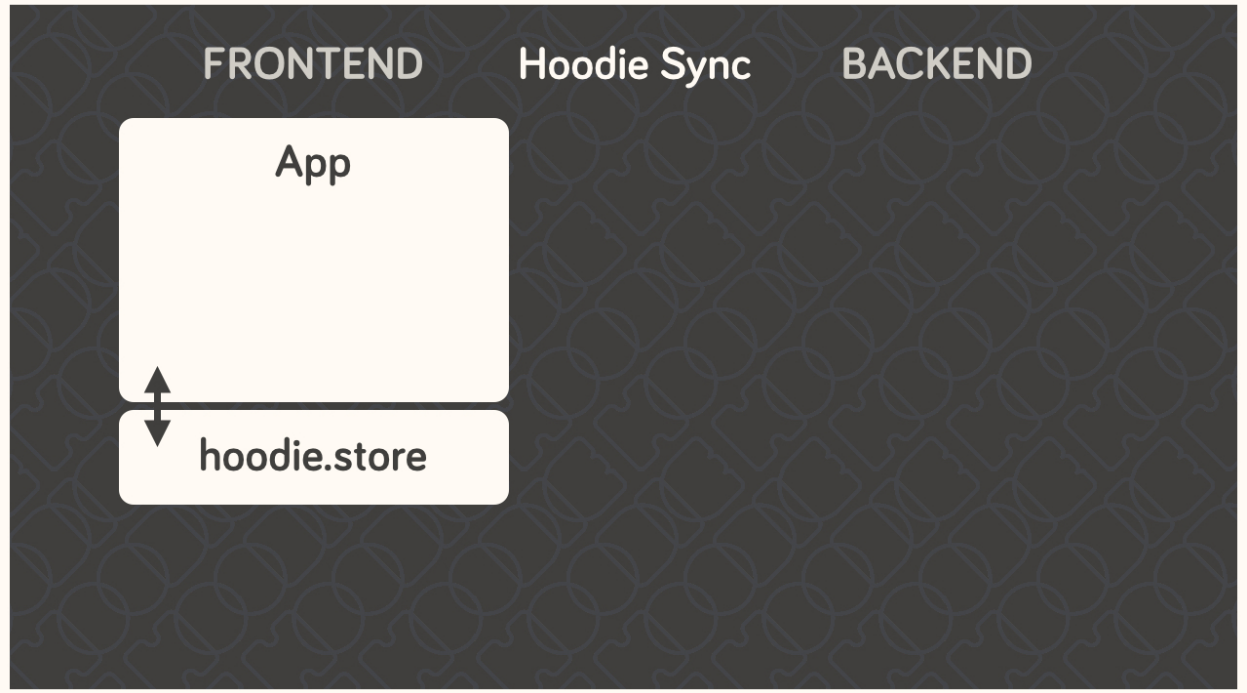
The app code only talks to the Hoodie frontend API, never directly to the server-side code, the database, or even the in-browser storage.

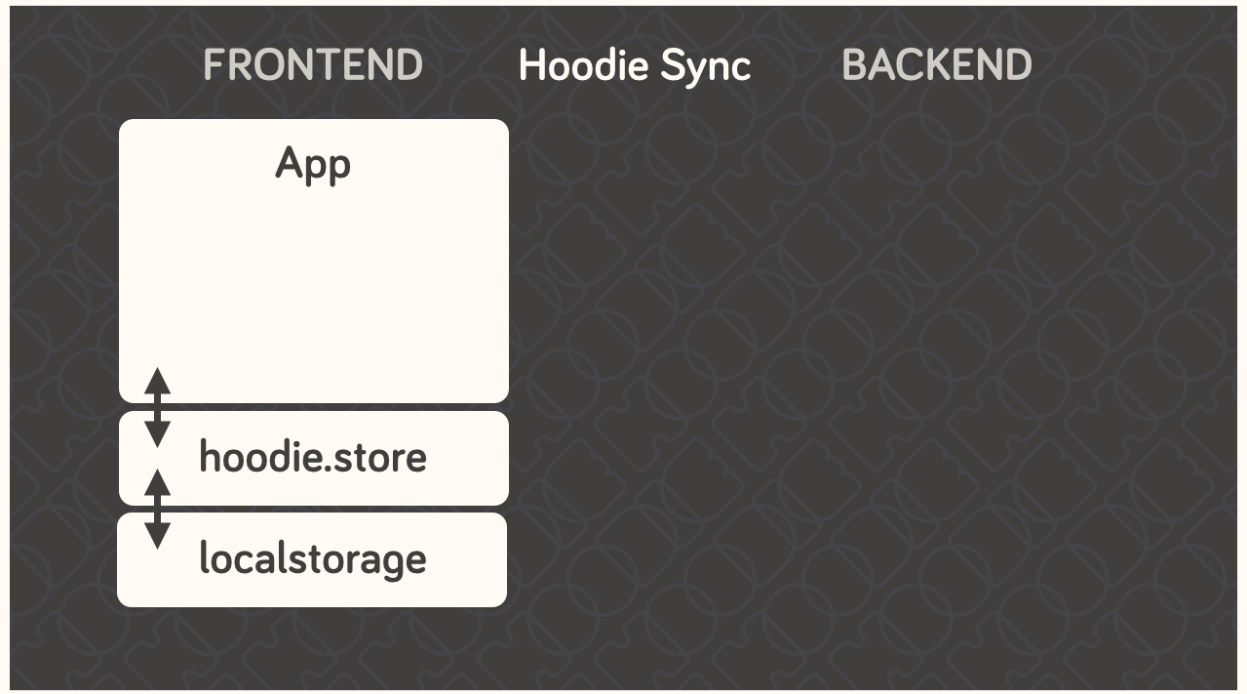
Hoodie uses PouchDB for storing data locally, which uses IndexedDb or WebSQL, whatever is available. Hoodie saves all data here first, before doing anything else. So if you're offline, your data is safely stored locally.

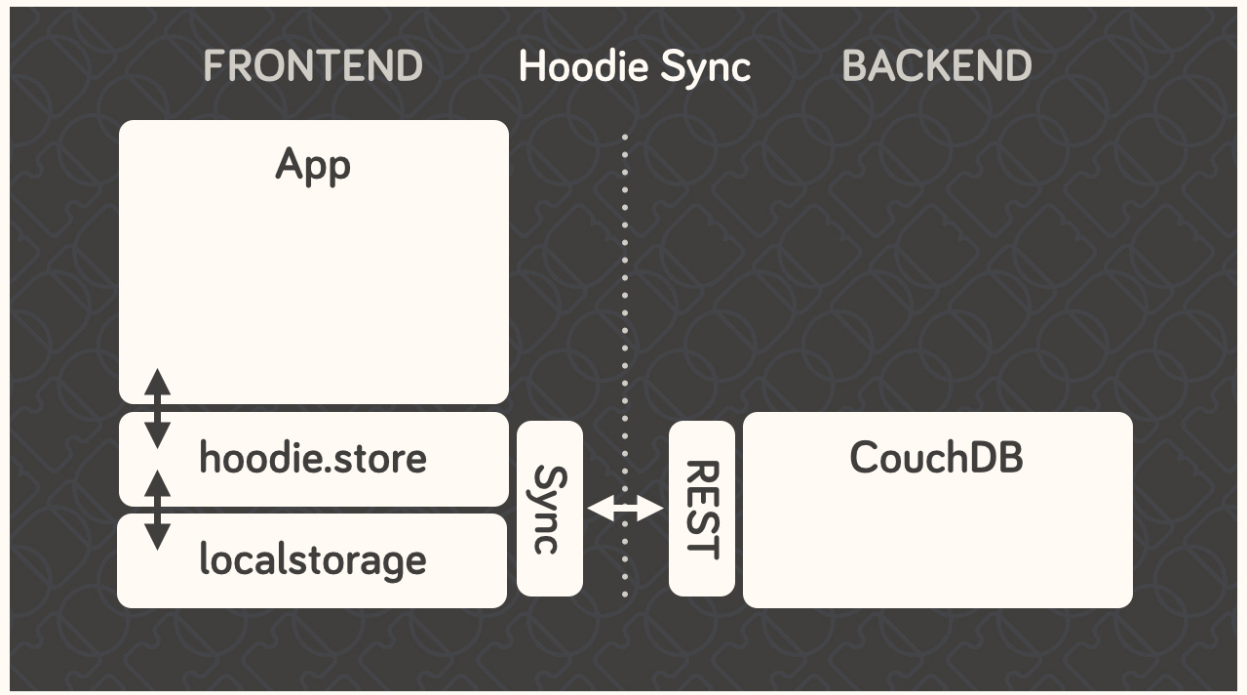
This, by itself, is already enough for an app. But if you want to save your data remotely or send an email, for example, you'll need a bit more.

Hoodie relies on CouchDB, the database that replicates. We use it to sync data back and forth between the server and the clients, which is something that CouchDB happens to be really good at.

A small aside: In CouchDB, each user has their own private database which only they can access, so all user data is


FRONTEND
Hoodie Sync
BACKEND
App
hoodie.store


FRONTEND
Hoodie Sync
BACKEND
App
hoodie.store
localstorage

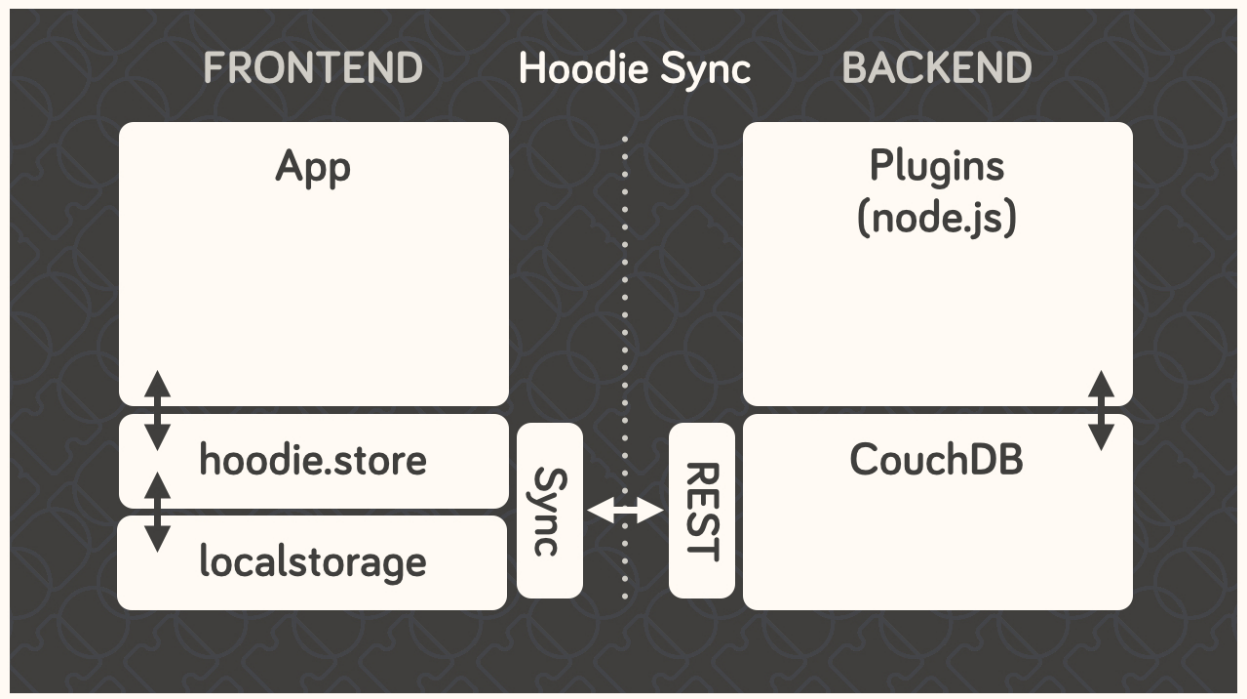


private by default. It can be shared to the public if the user decides to do so, but it can't happen by accident. This is why we'll often mention sharing and global data as a separate feature.

Behind the database, we have the actual server code in the form of a small node.js core with various plugins running alongside it. These then act upon the data in the CouchDB, which then replicates the changes back to the clients.

So Hoodie does **client database server** instead of the traditional **client server database**, and this is where many of its superpowers come from.

The clever bit is indicated by the dotted line in the middle; the connection between clients and server can be severed at any time without breaking the system. Frontend and backend never talk directly to each other. They only leave each other messages and tasks. It's all very loosely-coupled and event-based, and designed for eventual consistency.


FRONTEND
Hoodie Sync
BACKEND
App
Plugins
(node.js)
hoodie.store
localstorage
Sync
REST
CouchDB

CHAPTER 9

Architecture

After installing hoodie, `npm start` will run cli/index.js which reads out the configuration from all the different places using the rc package, then passes it as options to `server/index.js`, the Hoodie core hapi plugin.

In server/index.js, the passed options are merged with defaults and parsed into configuration for the Hapi server. It passes the configuration on to‘hoodie-server <https://github.com/hoodiehq/hoodie-server#readme>‘__, which combines the core server modules. It also bundles the Hoodie client on first request to `/hoodie/client.js` and passes in the configuration for the client. It also makes the app's `public` folder accessible at the `/` root path, and Hoodie's Core UIs at `/hoodie/admin`, `/hoodie/account` and `/hoodie/store`.

Hoodie uses CouchDB for data persistence. If `options.dbUrl` is not set, it falls back to PouchDB.

Once all configuration is taken care of, the internal plugins are initialised (see server/plugins/index.js). We define simple Hapi plugins for logging and for serving the app's public assets and the Hoodie client.

Once everything is setup, the server is then started at the end of cli/start.js and the URL where hoodie is running is logged to the terminal.

Modules

Hoodie is a server built on top of hapi with frontend APIs for account and store related tasks. It is split up in many small modules with the goal to lower the barrier to new code contributors and to share maintenance responsibilities.

1. server

   Hoodie's core server logic as hapi plugin. It integrates Hoodie's server core modules: account-server, store-server

   (a) account-server

   Hapi plugin that implements the Account JSON API routes and exposes a corresponding API at `server.plugins.account.api.*`.

   (b) store-server

   Hapi plugin that implements CouchDB's Document API. Compatible with CouchDB and PouchDB for persistence.

2. client

   Hoodie's front-end client for the browser. It integrates Hoodie's client core modules: account-client, store-client, connection-status and log

   (a) account-client

Client for the Account JSON API. It persists session information on the client and provides front-end friendly APIs for things like creating a user account, confirming, resetting a password, changing profile information, or closing the account.

(b) store-client

Store client for data persistence and offline sync. It combines pouchdb-hoodie-api and pouchdb-hoodie-sync.

i. pouchdb-hoodie-api

PouchDB plugin that provides simple methods to add, find, update and remove data.

ii. pouchdb-hoodie-sync

PouchDB plugin that provides simple methods to keep two databases in sync.

(c) connection-status

Browser library to monitor a connection status. It emits `disconnect` & `reconnect` events if the request status changes and persists its status on the client.

(d) log

JavaScript library for logging to the browser console. If available, it takes advantage of CSS-based styling of console log outputs.

5. admin

Hoodie's built-in Admin Dashboard, built with Ember.js

(a) admin-client

Hoodie's front-end admin client for the browser. Used in the Admin Dashboard, but can also be used standalone for custom admin dashboard.

CHAPTER 10

Files & Folders

package.json

TO BE DONE: Describe package.json file

README.md

TO BE DONE: Describe README file

.hoodie/

TO BE DONE: Describe .hoodie/ folder (caching of bundled client, data stored by PouchDB, ...)

hoodie/

TO BE DONE: Describe hoodie/ folder, extending app with hoodie/server/indexjs and hoodie/client/index.js.

public/

When you open your app in the browser you will see Hoodie's default page telling you that your app has no **public/** folder. So let's create it

```
mkdir public
touch public/index.html
```

Now edit the **public/index.html** file and pass in the following content.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My Hoodie App</title>
  </head>
  <body>
```

```
    <h1>My Hoodie App</h1>

    <script src="/hoodie/client.js"></script>
  </body>
</html>
```

You need to stop the server now (**ctrl + c**) and start it again. If you reload your app in your browser, you will now see your HTML file.

The only line interesting for us is t

CHAPTER 11

Requirements

Before you start working with Hoodie, here's what you need to know regarding your development/server environment and the browsers Hoodie will run in.

System Requirements for Hoodie Server

- Mac OSX
- Windows 7 and up
- Linux (Ubuntu, Fedora 19+)

Browser Compatibilities (all latest stable)

- Firefox (29+)
- Chrome (34+)
- Desktop Safari (7+)
- Internet Explorer 10+
- Opera (21+)
- Android 4.3+
- iOS Safari (7.1+)

Important: This list is currently based on PouchDB's requirements, since Hoodie is using PouchDB for its in-browser storage.

CHAPTER 12

Glossary

CouchDB

CouchDB is a non-relational, document-based database that replicates, which means it's really good at syncing data between multiple instances of itself. All data is stored as JSON, all indices (queries) are written in JavaScript, and it uses regular HTTP as its API.

PouchDB

PouchDB is an in-browser datastore inspired by CouchDB. It enables applications to store data locally while offline, then synchronize it with CouchDB.

hapi

hapi is a rich framework for building applications and services, enabling developers to focus on writing reusable application logic and not waste time with infrastructure logic. You can load hoodie as a hapi plugin to use it in your existing hapi application.

Users

Hoodie isn't a CMS, but a backend for web apps, and as such, it is very much centered around users. All of the offline and sync features are specific to each individual user's data, and each user's data is encapsulated from that of all others by default. This allows Hoodie to easily know what to sync between a user's clients and the server: simply all of the user's private data.

Private User Store

Every user signed up with your Hoodie app has their private little database. Anything you do in the **hoodie.store** methods stores data in here.